Federation Feeder Documentation

Release 0.9.4

Leif Johansson

Jan 29, 2018

Contents

Author *Leif Johansson <leifj@sunet.se>*

Release 0.9.4

pyFF is a simple but reasonably complete SAML metadata processor. It is intended to be used by anyone who needs to aggregate, validate, combine, transform, sign or publish SAML metadata.

Possible usecases include running an federation aggregator, filtering metadata for use by a discovery service, generating reports from metadata (eg certificate expiration reports), transforming metadata to add custom elements.

pyFF supports producing and validating digital signatures on SAML metadata using the pyXMLSecurity package which in turn supports using PKCS#11-modules - notoriously difficult to achieve using other tools.

pyFF is not a SAML metadata registry. If you need one of those have a look at the PEER project (also on pypi).

CHAPTER 1

Installation

1.1 Before you install

Make sure you have a reasonably modern python. pyFF is developed using 2.7 but 2.6 should work just fine. It is recommended that you install pyFF into a virtualenv but there are two ways: with or without site packages.

For both methods start by installing a few basic OS packages. Here we illustrate with commands for a debian/ubuntu install:

```
# apt-get install build-essential python-dev libxml2-dev libxslt1-dev libyaml-dev
```

and if you're on a centos system (or other yum-based systems):

```
# yum install python-devel  libxml2-devel libxslt-devel libyaml-devel
# easy_install pyyaml # bug in pip install pyyaml
# yum install make gcc kernel-devel kernel-headers glibc-headers
```

If you want to use OS packages instead of python packages from pypi then consider also installing the following packages before you begin:

```
# apt-get install python-lxml python-yaml python-eventlet python-setuptools
```

1.1.1 With Sitepackages

This method re-uses existing OS-level python packages. This means you'll have fewer worries keeping your python environment in sync with OS-level libraries.

```
# apt-get install python-virtualenv
# mkdir -p /opt/pyff
# virtualenv /opt/pyff
```

Choose this method if you want the OS to keep as many of your packages up to date for you.

1.1.2 Without Sitepackages

This method keeps everything inside your virtualenv. Use this method if you are developing pyFF or want to run multiple python-based applications in parallell without having to worry about conflicts between packages.

```
# apt-get install python-virtualenv
# mkdir -p /opt/pyff
# virtualenv /opt/pyff --no-site-packages
```

Choose this method for maximum control - ideal for development setups.

1.2 Installing

Now that you have a virtualenv, its time to install pyFF into it. Start by activating your virtualenv:

```
# . /opt/pyff/bin/activate
```

Next install pyFF:

```
# pip install pyFF
```

This will install a bunch of dependencies and compile bindings for both lxml, pyyaml aswell as pyXMLSecurity. This may take some time to complete. If there are no errors and if you have the *pyff* binary in your **$PATH** you should be done.

1.3 Upgrading

Unless you've made modifications, upgrading should be as simple as running

```
# . /opt/pyff/bin/activate
# pip install -U pyff
```

This should bring your virtualenv up to the latest version of pyff and its dependencies. You probably need to restart pyffd manually though.

CHAPTER 2

Using pyFF

pyFF has two command-line tools: pyff and pyffd.

```
# pyff --loglevel=INFO pipeline.fd [pipeline2.fd]
# pyffd --loglevel=INFO pipeline.fd [pipeline2.fd]
```

pyff operates by setting up and running "pipelines". Each pipeline starts with an empty "active repository" - an in-memory representation of a set of SAML metadata documents - and an empty "working document" - a subset of the EntityDescriptor elements in the active repository.

The pyffd tool starts a metadata server with an HTTP-based interface for viewing and downloading metadata. The HTTP interface can produce *XML*, *HTML* and *JSON* output (as well as other formats with a bit of configuration) and implements the MDX specification for online SAML metadata query.

Pipeline files are *yaml* documents representing a list of processing steps:

```
- step1
- step2
- step3
```

Each step represents a processing instruction. pyFF has a library of built-in instructions to choose from that include fetching local and remote metadata, xslt transforms, signing, validation and various forms of output and statistics.

Processing steps are called pipes. A pipe can have arguments and options:

```
- step [option]*:
    - argument1
    - argument2
    ...

- step [option]*:
    key1: value1
    key2: value2
    ...
```

Typically options are used to modify the behaviour of the pipe itself (think macros), while arguments provide runtime data to operate on.

Documentation for each pipe is in the `pyff.pipes.builtins` Module. Also take a look at the *Examples*.

CHAPTER 3

Examples

Examples are king.

3.1 Example 1 - A simple pull

Fetch SWAMID metadata, split it up into EntityDescriptor elements and store each as a separate file in /tmp/swamid.

```
- load:
     - http://mds.swamid.se/md/swamid-2.0.xml
- select
- publish: "/tmp/swamid-2.0.xml"
- stats
```

This is a simple example in 3 steps: load, select, store and stats. Each of these commands operate on a metadata repository that starts out as empty. The first command (load) causes a URL to be downloaded and the SAML metadata found there is stored in the metadata repository. The next command (select) creates an active document (which in this case consists of all EntityDescriptors in the metadata repository). Next, publish is called which causes the active document to be stored in an XML file. Finally the stats command prints out some information about the metadata repository.

This is essentially a 1-1 operation: the metadata loaded is stored in a local file. Next we'll look at a more complex example that involves filtering and transformation.

3.2 Example 2 - Grab the IdPs from edugain

Grab edugain metadata, select the IdPs (using an XPath expression), run it through the built-in 'tidy' XSL stylesheet (cf below) which cleans up some known problems, sign the result and write the lot to a file.

```
- load:
   - http://mds.edugain.org edugain-signer.crt
- select: "http://mds.edugain.org!//md:EntityDescriptor[md:IDPSSODescriptor]"
- xslt:
    stylesheet: tidy.xsl
- finalize:
    cacheDuration: PT5H
    validUntil: P10D
- sign:
    key: sign.key
    cert: sign.crt
- publish: /tmp/edugain-idp.xml
- stats
```

In this case the select (which uses an xpath in this case) picks the EntityDescriptors that contain at least one IDPSSODescriptor - in other words all IdPs. The xslt command transforms the result of this select using an xslt transformation. The finalize command sets cacheDuration and validUntil (to 10 days from the current date and time) on the EntitiesDescriptor element which is the result of calling select. The sign command performs an XML-dsig on the EntitiesDescriptor.

For reference the 'tidy' xsl is included with pyFF and looks like this:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:shibmeta="urn:mace:shibboleth:metadata:1.0"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
        xmlns:xi="http://www.w3.org/2001/XInclude"
                xmlns:shibmd="urn:mace:shibboleth:metadata:1.0">

  <xsl:template match="@ID"/>
  <xsl:template match="@validUntil"/>
  <xsl:template match="@cacheDuration"/>

  <xsl:template match="text()|comment()|@*">
    <xsl:copy/>
  </xsl:template>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

3.3 Example 3 - Use an XRD file

Sometimes it is useful to keep metadata URLs and signing certificates used for validation in a separate file and pyFF supports XRD-files for this purpose. Modify the previous example to look like this:

```
- load:
   - links.xrd
- select: "!//md:EntityDescriptor[md:IDPSSODescriptor]"
- xslt:
    stylesheet: tidy.xsl
```

```
- sign:
    key: sign.key
    cert: sign.crt
- publish: /tmp/idp.xml
- stats
```

Note that in this case the select doesn't include the http://mds.edugain.org prefix before the '!'-sign. This causes the xpath to operate on all source URLs, rather than just the single source http://mds.edugain.org . It would have been possible to call select with multiple arguments, each using a different URL from the file links.xrd which contains the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<XRDS xmlns="http://docs.oasis-open.org/ns/xri/xrd-1.0">
    <XRD>
        <Subject>http://mds.swamid.se/md/swamid-2.0.xml</Subject>
        <Link rel="urn:oasis:names:tc:SAML:2.0:metadata" href="http://mds.swamid.se/
↪md/swamid-2.0.xml">
            <Title>SWAMID</Title>
            <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:X509Data>
                    <ds:X509Certificate>
                    MIIFyzCCA7OgAwIBAgIJAI9LJsUJXDMVMA0GCSqGSIb3DQEBCwUAMHwxCzAJBgNV
                    BAYTAlNFMRIwEAYDVQQIDAlTdG9ja2hvbG0xEjAQBgNVBAcMCVN0b2NraG9sbTEO
                    MAwGA1UECgwFU1VORVQxDzANBgNVBAsMBlNXQU1JRDEkMCIGA1UEAwwbU1dBTUlE
                    IG1ldGFkYXRhIHNpZ25lciB2Mi4wMB4XDTE2MTIwNjA5MjgyMFoXDTM2MTIwNjA5
                    MjgyMFowfDELMAkGA1UEBhMCU0UxEjAQBgNVBAgMCVN0b2NraG9sbTESMBAGA1UE
                    BwwJU3RvY2tob2xtMQ4wDAYDVQQKDAVTVU5FVDEPMA0GA1UECwwGU1dBTUlEMSQw
                    IgYDVQQDDBtTV0FNSUQgbWV0YWRhdGEgc2lnbmVyIHYyLjAwggIiMA0GCSqGSIb3
                    DQEBAQUAA4ICDwAwggIKAoICAQDQVw72PnIo9QIeV439kQnPcxZh/LddKw86eIU+
                    nMfl4TpjSIyqTu4KJSnXbJyqXg+jQj3RzE9BUblpGrR7okmQwOh2nh+5A6SmyTOR
                    p7VEVT/Zw0GNnQi9gAW7J8Cy+Gnok4LeILI5u43hPylNKAnvs1+bo0ZlbHM6U5jm
                    6MlO+lrYA9dZzoPQqoCQbr3OweAaq5g8H54HuZacpYa3Q2GnUa4v+xywjntPdSQU
                    RTAbWWyJl3cHctX5+8UnX8nGCaxoBZqNp9PcEopyYJX8O1nrLumBMqu9Uh6GW1nx
                    OHfKDLvUoykG3Dm704ENVs88KaJXB1qQNsjdlm14UI9XCZbHfnFVnQ53ehsGFMha
                    Bf/Abd6v2wnhBLH/RxEUlw347qSeokw+SdDTSdW8jOEBiSqP/8BUzpCcbGlgAsVO
                    NKUS0K7IB2Bb79YYhyMvmJl24BGtkX+VM/mv47dxOtfzNFCMtUcJ2Dluv0xJG8xI
                    ot7umx/kbMBLuq7WdWELZJrgpt2bb9sXtYBpuxtGCW5g7+U7MNN1aKCiCSfq09YH
                    qu2DsU7HHAxEcGFXBiepBliCwZ24WLQh53bA3rihaln7SjdapT9VuSTpCvytb9RX
                    rq39mVuHMXvWYOG20XTV0+8U2vnsjAwsy28xPAcrLWRWoZbRJ+RoGp6L3GACq+t+
                    HPIukwIDAQABo1AwTjAdBgNVHQ4EFgQUQ2iqKQV/mMZDeJDtLXvy0Bsn/BQwHwYD
                    VR0jBBgwFoAUQ2iqKQV/mMZDeJDtLXvy0Bsn/BQwDAYDVR0TBAUwAwEB/zANBgkq
                    hkiG9w0BAQsFAAOCAgEAHviIAfS8viUN8Qk//U1p6Z1VK5718NeS7uqabug/SwhL
                    Vxtg/0x9FPJYf05HXj4moAf2W1ZLnhr0pnEPGDbdHAgDC672fpaAV7DO95d7xubc
                    rofR7Of2fehYSUZbXBWFiQ+xB5QfRsUFgB/qgHUolgn+4RXniiBYlWe6QJVncHx+
                    FtxD+vh1l5rLNkJgJLw2Lt3pbemSxUvv0CJtnK4jt2y95GsWGu1uSsVLrs0PR1Lj
                    kuxL6zZH4Pp9yjRDOUhbVYAnQ017mdcjvHYtp7c4GIWgyaBkDoMtU6fAt70QpeGj
                    XhecXk7Llx+oYNdZn14ZdFPRGMyAESLrT4Zf9M7QS3ypnWn/Ux0SwKWbnPUeRVbO
                    VZZ+M0jmdYK6o+UU5xH3peRWSJIjjRaKjbVlW5GgHwGFmQc/LN+va2jjThRsQWWt
                    zEwObijedInQ6wfL/VzFAwlWWoDAzKK9qnK4Rf3ORKkvhKrUa//2OYnZD0kHtHiC
                    OL+iFRLtJ/DQP5iZAF+M1Hta7acLmQ8v7Mn1ZR9lyDWzFx57VOKKtJ6RAmBvxOdP
                    8cIgBNvLAEdXh2knOLqYU/CeaGkxTD7Y0SEKx6OxEEdafba//MBkVLt4bRoLXts6
                    6JY25FqFh3eJZjR6h4W1NW8KnBWuy+ITGfXxoJSsX78/pwAY+v32jRxMZGUi1J4=
                    </ds:X509Certificate>
                </ds:X509Data>
          </ds:KeyInfo>
        </Link>
    </XRD>
    <XRD>
```

```
        <Subject>https://incommon.org</Subject>
        <Link rel="urn:oasis:names:tc:SAML:2.0:metadata" href="http://md.incommon.org/
↪InCommon/InCommon-metadata.xml">
            <Title>InCommon Metadata (main aggregate)</Title>
            <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:X509Data>
                    <ds:X509Certificate>
     MIIDgTCCAmmgAwIBAgIJAJRJzvdpkmNaMA0GCSqGSIb3DQEBCwUAMFcxCzAJBgNV
     BAYTAlVTMRUwEwYDVQQKDAxJbkNvbW1vbiBMTEMxMTAvBgNVBAMMKEluQ29tbW9u
     IEZlZGVyYXRpb24gTWV0YWRhdGEgU2lnbmluZyBLZXkwHhcNMTMxMjE2MTkzNDU1
     WhcNMzcxMjE4MTkzNDU1WjBXMQswCQYDVQQGEwJVUzEVMBMGA1UECgwMSW5Db21t
     b24gTExDMTEwLwYDVQQDDChJbkNvbW1vbiBGZWRlcmF0aW9uIE1ldGFkYXRhIFNp
     Z25pbmcgS2V5MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0Chdkrn+
     dG5Zj5L3UIw+xeWgNzm8ajw7/FyqRQ1SjD4Lfg2WCdlfjOrYGNnVZMCTfItoXTSp
     g4rXxHQsykeNiYRu2+02uMS+1pnBqWjzdPJE0od+q8EbdvE6ShimjyNn0yQfGyQK
     CNdYuc+75MIHsaIOAEtDZUST9Sd4oeU1zRjV2sGvUd+JFHveUAhRc0b+JEZfIEuq
     /LIU9qxm/+gFaawlmojZPyOWZ1JlswbrrJYYyn10qgnJvjh9gZWXKjmPxqvHKJcA
     TPhAh2gWGabWTXBJCckMe1hrHCl/vbDLCmz0/oYuoaSDzP6zE9YSA/xCplaHA0mo
     C1Vs2H5MOQGlewIDAQABo1AwTjAdBgNVHQ4EFgQU5ij9YLU5zQ6K75kPgVpyQ2N/
     lPswHwYDVR0jBBgwFoAU5ij9YLU5zQ6K75kPgVpyQ2N/lPswDAYDVR0TBAUwAwEB
     /zANBgkqhkiG9w0BAQsFAAOCAQEAaQkEx9xvaLUt0PNLvHMtxXQPedCPw5xQBd2V
     WOsWPYspRAOSNbU1VloY+xUkUKorYTogKUY1q+uh2gDIEazW0uZZaQvWPp8xdxWq
     Dh96n5US06lszEc+Lj3dqdxWkXRRqEbjhBFh/utXaeyeSOtaX65GwD5svDHnJBcl
     AGkzeRIXqxmYG+I2zMm/JYGzEnbwToyC7yF6Q8cQxOr37hEpqz+WN/x3qM2qyBLE
     CQFjmlJrvRLkSL15PCZiu+xFNFd/zx6btDun5DBlfDS9DG+SHCNH6Nq+NfP+ZQ8C
     GzP/3TaZPzMlKPDCjp0XOQfyQqFIXdwjPFTWjEusDBlm4qJAlQ==
                    </ds:X509Certificate>
                </ds:X509Data>
          </ds:KeyInfo>
        </Link>
    </XRD>
</XRDS>
```

The structure of the file should be fairly self-evident. Only links with @rel="urn:oasis:names:tc:SAML:2.0:metadata" will be parsed. If a KeyInfo with a X509Certificate element (usual base64-encoded certificate format) then this certificate is used to validate the signature on the downloaded SAML metadata. Note that while 'load' supports validation based on certificate fingerprint the XRD format does not and you will have to include Base64-encoded certificates if you want validation to work.

3.4 Example 4 - Sign using a PKCS#11 module

Fetch SWAMID metadata (and validate the signature using a certificate matching the given SHA1 fingerprint), select the Identity Providers, tidy it up a bit and sign with the key with the label 'signer' in the PKCS#11 module /usr/lib/libsofthsm.so. If a certificate is found in the same PKCS#11 object, that certificate is included in the Signature object.

```
- load:
   - http://mds.swamid.se/md/swamid-2.0.xml␣
↪A6:78:5A:37:C9:C9:0C:25:AD:5F:1F:69:22:EF:76:7B:C9:78:67:67:3A:AF:4F:8B:EA:A1:A7:6D:A3:A8:E5:85
- select: "!//md:EntityDescriptor[md:IDPSSODescriptor]"
- xslt:
    stylesheet: tidy.xsl
- sign:
    key: pkcs11:///usr/lib/libsofthsm.so/signer
- publish: /tmp/idp.xml
- stats
```

Running this example requires some preparation. Run the ‘p11setup.sh’ script in the examples directory. This results in a SoftHSM token being setup with the PIN ‘secret1’ and SO_PIN ‘secret2’. Now run pyFF (assuming you are using a unix-like environment).

```
# env PYKCS11PIN=secret1 SOFTHSM_CONF=softhsm.conf pyff --loglevel=DEBUG p11.fd
```

3.5 Example 5 - MDX

Running an MDX server is pretty easy using pyFF. Lets start with the links.xrd file (cf example above) and add this simple pipeline.

```
- when update:
    - load:
       - links.xrd
    - break
- when request:
    - select
    - pipe:
        - when accept application/xml:
             - xslt:
                 stylesheet: tidy.xsl
             - first
             - finalize:
                cacheDuration: PT5H
                validUntil: P10D
             - sign:
                 key: sign.key
                 cert: sign.crt
             - emit application/xml
             - break
        - when accept application/json:
             - xslt:
                 stylesheet: discojson.xsl
             - emit application/json:
             - break
```

The big difference here are the two when commands. They are used to select between the two main entrypoints for the pyFF server: the update flow and the request flow. The update flow is run repeatedly and is usually used for updating the internal metadata repository.

The request flow is called every time an MDX request is submitted. The internal when statements are used to provide basic content negotiation for the MDX request. Content negotiation is based both on the Accept header and on the extension (suffix) on the URL - ending a resource with ‘.json’ selects application/json, etc and overrides the Accept header.

The only new commands here are emit, break and first. The emit command transforms the result into the appropriate output format (UTF-8 encoded text), the break terminates the pipeline. The first command strips the outer EntitiesDescriptor if only a single EntityDescriptor is present in the active document which is consistent with expected behaviour for the MDX protocol.

The behaviour of the select command in the request pipeline is a bit different: the select operates on a query fed to the request pipeline from the HTTP server that runs the command. This is called implicit select.

Now start pyffd:

```
# pyffd -f --loglevel=DEBUG -p /var/run/pyffd.pid mdx.fd
```

This should start pyffd in the foreground. If you remove the `-f` pyFF should daemonize. For running pyFF in production I suggest something like this:

```
# pyffd --loglevel=INFO --log=syslog:auth --frequency=300 -p /var/run/pyffd.pid --
↪dir=`pwd` -H<ip> -P80 mdx.fd
```

This starts pyff on the interface <ip>:80 and uses the current directory as the working directory. If you leave out –dir then pyffd will change directory to $HOME of the current user which is probably not what you want. In this case logging is done through syslog (the auth facility) and with log level INFO. The refresh-rate is set to 300 seconds so at minimum your downstream feeds will be refreshed that often.

CHAPTER 4

Extending pyFF

Not much here yet - come back later or UTSL

CHAPTER 5

Frequenty Asked Questions

Q: I get 'select is empty' but I know my xpath should match. What is wrong?

A: You've probably forgotten to include namespaces in your xpath expression. The expression "//EntityDescriptor" won't match anything - //md:EntityDescriptor" is what you want.

The pyFF logo is the chemical symbol for sublimation - a process by which elements are transitioned from solid to gas without becoming liquids.